…Data collection stays basically the same, so I'm just re-posting this here.

```
suppressPackageStartupMessages(library(rStrava))
suppressPackageStartupMessages(library(lubridate))
suppressPackageStartupMessages(library(gganimate))
suppressPackageStartupMessages(library(scales))
# Get the data from the API
app_name <- ''
app_client_id  <- ''
app_secret <- ''

# create the authentication token
stoken <- httr::config(token = strava_oauth(app_name, app_client_id,
app_secret, app_scope="activity:read_all"))
act.list <- get_activity_list(stoken)
act.df <- compile_activities(act.list)
act.df <- act.df[,c("start_date_local", "distance", "type")]
```

Alright, we're all set for the modifications to the code now. Remember that you can also plug in your own dataframe as long as it fits the structure of the API-generated one.

Code modifications

Let's say we still want to run 500 km in one year, but now we want to compare several years - let's choose the last 3 years:

```
goal <- 500
sports <- "Run"
selected.year <- c(2018, 2019, 2020)
```

We are subsetting and aggregating the data. Just as a reminder: We have to aggregate in case we have several activities at one day (like one run split in two activities).

```
select.df <- act.df[act.df$type %in% sports &
                      year(act.df$start_date_local) %in%
selected.year,]

select.df$date <- as.Date(date(select.df$start_date_local))
select.df <- aggregate(distance ~ date, data = select.df, FUN = sum)
select.df <- select.df[order(select.df$date),]
```

The dataframe with every date (not only the ones with recorded activities) is still called `year.df`. We have to make sure that this is done for each year in the dataset. I'm using an `sapply()` for this.

```
year.df <- data.frame(
  date = unlist(sapply(unique(year(select.df$date)), USE.NAMES = F, FUN
= function (yr) {
    as.character(seq(as.Date(paste0(yr, "-01-01")),
        as.Date(paste0(yr, "-12-31")),
        by = "day"))
  }))
```

```
)
year.df$date <- as.Date(year.df$date)
```

Now, we are

- integrating the data from `select.df` into `year.df` with a call to `merge()`
- ordering by date,
- replacing `NA`s with zeros in the `distance` column
- creating a column with the day of year (with the `yday()` function from {lubridate}) because this will be our new x-axis variable. We have to do this because we want to see all years in the dataset develop "next to each other". The day of year is the same for all years (with a minor exception: in leap years, days will be shifted by 1 starting from March 1st - this won't be visible in the plot though)
- creating a column with the year which we will use later for the `col` and `group` aesthetic of the plot.

```
year.df <- merge(year.df, select.df, by = "date", all.x = T, all.y = F)
year.df <- year.df[order(year.df$date),]
year.df$distance <- ifelse(is.na(year.df$distance), 0, year.df$distance)
year.df$day <- yday(year.df$date)
year.df$year <- year(year.df$date)
```

The next call makes use of some {dplyr} functions and piping. Although I have been sceptical towards the "tidyverse" in the beginning, I am now using some functionality whenever it has huge advantages compared to base R functionality. The following code chunk shows such an advantage, I think.

- We're grouping `year.df` by year.
- We are cumulating the distances within each group (because cumulated distance has to start with 0 at the beginning of the year)
- We are ungrouping and converting the tibble back to dataframe (because otherwise it would break later if we would keep the tibble).

```
suppressPackageStartupMessages(library(dplyr))
year.df %>% group_by(year) %>%
  mutate(cum.distance = round(cumsum(distance))) %>%
  ungroup() %>% as.data.frame() -> year.df
```

Now, we are finally computing all the other statistics used in the plot (see the previous post for further explanations). I am keeping *all* the statistics although currently not all of them are being used. I got rid of the large label in the lower right corner because I don't have a good idea to still keep it simple and short with 3 years.

```
year.df$remaining.distance <- goal - year.df$cum.distance
year.df$days.till.end <- as.numeric(as.Date(paste0(year.df$year,
"-12-31")) -
                                      year.df$date)

year.df$dist.per.day.to.goal <- year.df$remaining.distance /
year.df$days.till.end
year.df$dist.per.week.to.goal <- year.df$dist.per.day.to.goal * 7

year.df$dist.per.day.to.goal <- round(year.df$dist.per.day.to.goal, 1)
```

```
year.df$dist.per.week.to.goal <- round(year.df$dist.per.week.to.goal,
1)

year.df[year.df$dist.per.day.to.goal < 0, "dist.per.day.to.goal"] <- 0
year.df[year.df$dist.per.week.to.goal < 0, "dist.per.week.to.goal"] <-
0
year.df[year.df$remaining.distance < 0, "remaining.distance"] <- 0

year.df[year.df$days.till.end == 0, "dist.per.day.to.goal"] <- "-"
year.df[year.df$days.till.end == 0, "dist.per.week.to.goal"] <- "-"

current.day <- yday(Sys.Date())
```

Plotting and animating

The only important change here is that the x-axis is defined by the day of the year now. I am including major and minor breaks at the beginning of the months. The x-axis labels are given for every second month.

```
p <- ggplot(year.df, aes(x = day, y = cum.distance, col = factor(year),
group = year)) +
  geom_line() +
  geom_hline(yintercept = goal, col = "red", lwd = 1) +
  geom_vline(xintercept = current.day, col = alpha("grey", .5), lty =
"dashed") +
  geom_segment(x = 1, y = 0, xend = 365, yend = goal, lty = "dashed",
col = "red") +
  geom_segment(aes(x = day, y = cum.distance, col = factor(year), group
= year),
               xend = 365,
               yend = goal, lty = "dotted") +
  geom_point(size = 2) +
  geom_text(aes(day + 7,
                label = paste0(year, ": ", round(cum.distance))),
            hjust = 0, size = 4) +
  coord_cartesian(clip = 'off') +
  scale_x_continuous(breaks = c(32, 91, 152, 213, 274, 335),
                     labels = c("Feb", "Apr", "Jun", "Aug", "Oct",
"Dec"),
                     minor_breaks = c(1, 60, 121, 182, 244, 305, 365))
+
  transition_reveal(day) +
  guides(col = F) +
  labs(x = "Day of year", y = "km completed") +
  theme_minimal() + theme(plot.margin = margin(5.5, 60, 5.5, 5.5))
```

Aaaaand… animating the whole thing!

```
animate(p, fps = 60, duration = 30, width = 1440, height = 900, res =
200, end_pause = 25,
        renderer = av_renderer(file = "runs-3yr.mp4"))
```